07 - 10 September 2009

PROCEEDINGS

54. IWK

Internationales Wissenschaftliches Kolloquium
International Scientific Colloquium

Information Technology and Electrical Engineering - Devices and Systems, Materials and Technologies for the Future

Faculty of Electrical Engineering and Information Technology

Startseite / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=14089

TECHNISCHE UNIVERSITÄT
ILMENAU

Impressum

Herausgeber: Der Rektor der Technischen Universität Ilmenau
Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c.
Peter Scharff

Redaktion: Referat Marketing
Andrea Schneider

Fakultät für Elektrotechnik und Informationstechnik
Univ.-Prof. Dr.-Ing. Frank Berger

Redaktionsschluss: 17. August 2009

Technische Realisierung (USB-Flash-Ausgabe):
Institut für Medientechnik an der TU Ilmenau
Dipl.-Ing. Christian Weigel
Dipl.-Ing. Helge Drumm

Technische Realisierung (Online-Ausgabe):
Universitätsbibliothek Ilmenau
ilmedia
Postfach 10 05 65
98684 Ilmenau

Verlag:
Verlag ISLE, Betriebsstätte des ISLE e.V.
Werner-von-Siemens-Str. 16
98693 Ilmenau

ISBN (USB-Flash-Ausgabe): 978-3-938843-45-1
ISBN (Druckausgabe der Kurzfassungen): 978-3-938843-44-4

Startseite / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=14089

VIRTUAL AUDIO PROCESSORS FOR THE DEVELOPMENT OF DIGITAL AUDIO EQUIPMENT

S. Jaritz / T. Bender / H. Kahnt

University of Applied Sciences Jena
Department of Electrical Engineering and Information Technology
Carl-Zeiss-Promenade 2
07745 Jena, Germany
stefan.jaritz@fh-jena.de

ABSTRACT

The aim of presented method is to speed up the development of new audio devices and design reusable hardware and software. The reuse of the software is assured by defining a virtual audio processor with a specific set of instructions. The definition of a hardware abstraction layer provides the use of different hardware. The developer describes the new system in a platform independent language. A common communication protocol makes it possible to connect different types of hardware.

Index Terms— audio language, virtual audio processor, audio compiler, audio signal processing

1. INTRODUCTION

In the last years code generating tools are getting more and more popular. The benefits of them are more flexible codes, cross platform development, hardware flexibility and faster development processes. The gain of flexibility is bought by the cost of performance decrease and bigger code size. The development process is characterized by customized tools like compilers, interpreters, virtual machines and configuration scripts. Professional digital audio equipment is characterized by fast innovation. Increasing computing power, memory bandwidth and high speed interconnections offer the possibility to realize new and more complex audio devices. Most electronic active components such like μC's, DSP's and FPGA's have lifetimes below five years. Manufacturers have to deal with frequently changes of their product design. Often, redesign of the hardware requires modifications in software or - in bad case - a complete new software development. This inflicts heavy costs. The concept of the Virtual Audio Processor (VAP) is a new approach to improve hardware and software development.

2. OVERVIEW OF DESIGNING METHODS

2.1. Traditional method

The traditional method for creating new audio systems is the following:

1. Defining signal processing, input and output signals
2. Defining the user roles, user interface
3. Research for new algorithms, checking older projects for usable patterns
4. Software and Hardware development - using for every part the tool given by the manufacture (different compilers, libraries, message systems, etc.)
5. Quality assurance

This method requires an efficient development process and high level of expertise for all involved developers. Permanent communication between developers is required on each step of the process. An error at step 4 or 5 results in backtracking to steps 1 and 2. The communication between the system components is defined separately for every component (own protocol, arbitrage, etc.). This approach leads to highly specialized products and makes it difficult to extend or to combine audio components into large systems. Any changes of software or hardware can only be made by passing through the entire developing process.

2.2. Improved method using Matlab/Simulink

Matlab/Simulink is a powerful tool which allows the development of an audio component. Signal processing can be simulated and tested by means of a graphical user interface. A huge and tested library of functions speeds up the development process. Code generation is realized by the Linker to DSP. This approach is limited by the quality of the code generation. The Linker to DSP does not support all features of the listed DSPs and dispatching functions on multi-component systems.

2.3. Methods under development

2.3.1. CLAM: A framework for audio and music application development [1]

The CLAM framework is a powerful open source tool to develop new audio algorithms. It comes with a large library with audio signal processing functions. Using a platform independent structure makes it easy to port the code to new platforms. The most significant weakness of CLAM is that only the algorithm and not the whole device can be developed. Another disadvantage is the complete implementation in C++. For most micro controlers and DSPs a C++ compiler doesn't exist, and if it exists it produces in most cases inefficient and large codes.

2.3.2. RTPROC: A system for rapid real-time prototyping in audio signal processing [2]

This method is abstracting the signal processing into three modules. The driver module transforms all kinds of inputs to a defined standard. The signal processing is done by the algorithm module. The system is controlled by the host. The host is able to communicate with other devices and to set parameters of the algorithm.

2.3.3. Industrial experience using rule-driven retargetable code generation for multimedia applications [3]

This paper compares the efficiency of rule driven compilers for code generation. The algorithm is transformed to a code for a non existing virtual machine. In further steps this code is transformed to a platform optimized code. This article shows that it is possible that machine generated code is equal in terms of size and performance to hand coded code.

2.3.4. An audio virtual DSP for multimedia frameworks [4]

This article describes how an audio interpreter for MPEG-4 Structured Audio (SA) and Binary Format for Scenes (BIFS) can be created. It's called the "'SAINT VIRTUAL DSP"'. The DSP uses marcoinstructions and has no memory stack. The macroinstructions are used direct from the ALU of the machine. There is a scheduler witch dispatches the code to several processing units. The performance of the interpreter is 20%-30% lower to cross-compiled solutions. At the other hand the system is flexible for new applications.

3. THE METHOD OF THE VIRTUAL AUDIO PROCESSOR

This method consists of the following parts:

- Audio Language - Human readable code which describes a whole audio system including signal processing and routing, user interactions and interface, control and data exchange
- Audio Language Compiler - Check of syntax and grammar of the Audio Language Code and translation into a platform independent binary code
- Virtual Audio Processor (HAL, Message System, Debug-Log-System) - The HAL presents the interface between hardware and software. It consists of the complete set of processor functions which are accessible by the developer. The definition of a HAL is specialized for audio signal processing and guarantees the usage of the specific processor features. The defined message system enables distributed systems. By the introduction of a HAL the program code becomes independent of the hardware and therefore, it is reusable and can be organized in libraries. The same hardware can be used for different tasks. Also an old program can be executed on new hardware. The creation of an efficient HAL requires highly skilled programmers. However, this work has to be done only once for a given DSP or μC.

The trend of creating platform independent code is discussed in various articles (s.a. and [5],[6]). In common a performance loss to 30% is realistic [4]. This can be lowered by using marcofunctions, transforming the instruction to code (the decode step is saved) and an optimizer which optimizes the platform independent code.

3.1. Idea of abstracting device functions into a hardware abstraction layer

A device can be characterized by its functions. These functions can be divided into smaller functions. Repeating this process leads to the basic functions. This set of basic functions is provided by the so called hardware abstraction layer. The figure 1 demonstrates how the device is splitted to its functions (F). By abstracting them (A) to basic functions (BF), the device functions are mapped to HAL functions.

The set of the HAL functions can be divided into subsets. These include arithmetic, signal processing, debugging, logging/monitoring, user interaction and communication functions. The HAL doesn't implement any function. It gives only a definition for them. For each function a platform independent implementation is created. These implementations are put into a set called hardware emulation layer (HEL). Setting up a new audio processor can be understood as the process of merging HEL-functions and platform specific implementations of functions defined by the HAL. The HAL architecture makes it easy to use hardware specials like

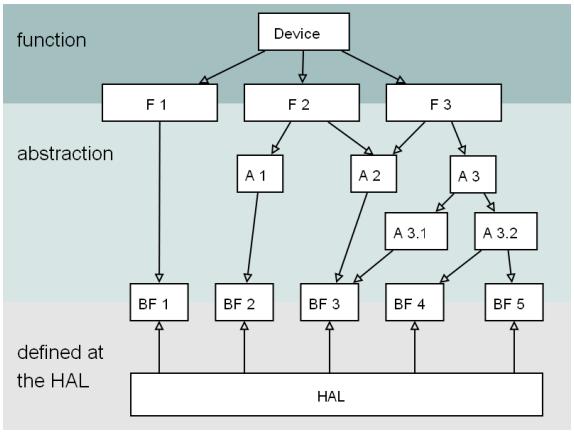

Fig. 1. Abstraction process

FFT's, sample rate converters, etc. By using the HEL function as reference to the user created implementations of the HAL functions they can be easily tested and verified. The use of a HAL makes it easy to react on changes in Hardware.

3.2. The Audio Language

The aim of the audio language is to enable the user to design a device by using human readable code (HRC). This code has a syntax and grammar. Both together are called the audio language. The audio language is a function orientated language with data types. The function orientation has two goals. First the code becomes more modular. Second the code can be organized in libraries which results in better reusability of it. The data types are called components. The components consist of structure and algorithms. The algorithm uses the structure for its work. The following example shows how this works. We got a digital 2nd order filter. The formula to calculate y(n) is given by the direct form II:

$$y(n) = (x + w_0) * K \tag{1}$$

$$w_0(n+1) = x * b_1 + w_1(n) - a_1 * (x + w_0(n)) \tag{2}$$

$$w_1(n+1) = x * b_2 - a_2 * (x + w_1(n)) \tag{3}$$

Using the direct form II for calculating this formula leads to two memory cells. It follows that the structure of the 2nd order filter has two memory cells w_0 and w_1 and five constants K, a_1, a_2, b_1 and b_2. The function of the processing algorithm is the formula for the direct form II. For the use of this digital filter, the coefficients have to be set. This is done by the init function. The data for the initialization, like sample frequency, cut-off frequency, Q - factor etc. are saved in the structure too. These data are human readable. The init function transforms it by its rules to the non human readable filter coefficients. The data structure is dived into three segments. There is the PD (= process data) segment for data, which is for often modified at runtime. The CD (constant data) segment is for constants and coefficients. The initial data are stored at the ID segment. This separation give more flexibility to the developer to use system benefits like two memory busses, different caches etc.. The processing function convolutes an input sample with the transfer function and returns an output sample. The component is a subsystem at the HAL (fig. 2).

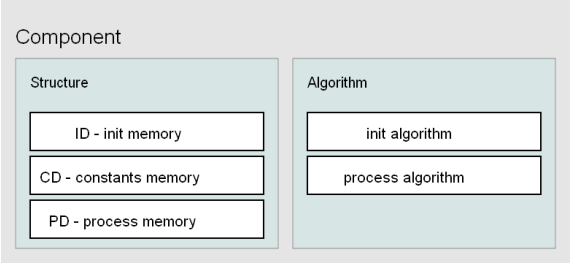

Fig. 2. Definition of a component

With this structure it is possible to use CPU build-in features like sample rate converters or FFT's for the algorithm. The standard HEL function is to be replaced by the special function. Another benefit is the memory organization. It is separated form the algorithm and can be optimized form the surrounding system. These things are hidden for the end user. This user writes a code like this:

```
globals
{
HP h1;
}

function "HPinit" [] ()
begin
locals
{
}
code
{
h1.fs = 44100; // set sample frequency to 44.1kHz
h1.fc = 1000; // cut off frequency set to 1kHz
h1.q = 0.707; // Q-factor
init (h1); // init filter
}
end

function "HPprocessing" [] ()
begin
locals
{
RATIONAL  x1;
}
code
{
getInput(x1,1,0); // read sample from channel 1
x1 = h1 * x1; // convolute with filter
setOutput(x1,4,0);// write to output channel 4
}
end
```

This code defines a high pass filter. The "HP init" function initializes it and the "HP processing" reads one sample form the inputs, passes it through the filter and writes the result to the output.

3.3. Transforming the Audio Language to a platform-independent code

The audio language compiler (ALC) checks the syntax. If there are errors the user gets some message. After the syntax check is done the compilers collects all functions used for the device and build an internal code. Now the translation starts. After recognizing the pattern the compiler transforms the human readable code (HRC) into the audio code (AC v1). In this step the compiler uses only the functions defined by the HAL. The result of this is a platform independent binary code. At the step where the code of the HRC functions is merged and at the translation to AC v1 the compiler is able to make optimizations. For testing the code and fixing the bugs a debugger is a nessary tool. The debugger can be understood as implementation of the HAL with the complete functions of the HEL. A debugger for the audio language has been created. As mentioned above, the virtual audio processor organizes the code and data into different segments:

- CS - code segment: the list with instructions is stored here
- PD - process data segment: data which are changed often at runtime are stored here
- CD - constant data segment: data like constants and coefficients are stored here
- ID - init data segment: data for initializing the components are stored here

The compiler generates the four segments and puts them into one package. Additional information for debugging purposes can be included in this package. Every hardware, running an audio processor, is able to load this package and execute it.

3.4. Localizing the code on a device

With the implementation of the HAL of the Virtual Audio Processor a real audio processor is created. The real audio processor includes a program loader, a message system and an interpreter which uses the functions of the HAL. The program loader transforms the AC v1 into a platform specific optimized code called AC v2. It performs several tasks:

1. memory organization
2. decoding function identifiers to function pointers
3. type conversations

3.4.1. Memory organization

The program loader can arrange these memory segments in a way that the hardware architecture is optimally used. For the ADSP series from Analog Devices the usage of the two memory busses (PD and CD) means a performance gain of 50% for memory accesses.

3.4.2. Decoding function identifiers to function pointers

Every HAL function has a unique identifier (id). When the code is interpreted a function has to be assigned to the id. Decoding function id's to function pointers results in a significant performance gain. The code is decoded before execution. This leads to a faster interpreter.

3.4.3. Type conversations

The representation of the integers on hardware as little, big or mid endian is different. The loader converts the variables at each system to the right representation at the hardware.

3.5. Clustering devices

Most hardware has special features. The ARM9 processors are ideal for communication and monitoring systems. The ADSP series form Analog Devices are powerful signal processors. A PC is good to use as remote control. In the past the communication protocol was specially designed for every device. By defining a common protocol old and new hardware can be easily connected. It's like a plug and play system. Using that system makes it much easier to reuse hardware. Also hardware upgrades can be easily applied to the whole device. Modular Hardware is archived by defined communication protocols.

3.6. Supported targets

The demo system runs on a ARM9 demo-board from taskit (AT91RM9200), ARM7, x86 and ADSP-21369 eval-board. The algorithms are written in ANSI-C (C89) and are portable to each device with a standard c-compiler. For the inputs and the communication extra drivers are necessary.

3.7. Supported OS

Several OS are supported. For the x86 platform windows, for ARM9 an embedded Linux, for ARM7 FreeRTOS is supported. Because of the platform independent source code it can be easily transferred to other OS.

3.8. The future of the Audio Language

There are new developments in two areas. The abstraction of user interfaces (UI) is finished. An implementation on different systems has to be done. The MIDI interface is going to be added to the system.

4. REFERENCES

[1] Xavier Amatriain, "CLAM: A Framework for Audio and Music Application Development," *IEEE Software*, pp. 82–85, 2007.

[2] Hauke Krueger and Peter Vary, "RTPROC: A System for Rapid Real-Time Prototyping in Audio Signal Processing," *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pp. 311–314, 2008.

[3] C. Liem, P. Paulin, M. Cornero, and A. Jerraya, "Industrial experience using rule-driven retargetable code generation for multimedia applications," *Eighth International Symposium on System Synthesis*, p. 60, sep 1995.

[4] G. Zoia and C. Alberti, "An audio virtual DSP for multimedia frameworks," *Acoustics, Speech, and Signal Processing, IEEE International Conference*, vol. 3, pp. 1421–1424, mar 2001.

[5] Tom Erkkinen, "Damals und heute - Generierung von Produktionscodes," *Elektronik embedded*, vol. 1, pp. 48, feb 2009.

[6] Klaus-Dieter Walter, "Hardware Unabhaengigkeit - der Schluessel zur guten Software," *Elektronik*, vol. 7, pp. 42, mar 2009.